1.2. Otros usos de asertos

- **Calcular que condiciones se han de cumplir para que al acabar un programa se cumplan ciertas propiedades.**

Ejemplo: Dado el siguiente programa que calcula la media de los elementos de un vector:

```
media := (Σα:1 <= α <= N: v[α])/N
```

Queremos saber las condiciones que se deben cumplir para que el resultado final sea positivo. Para ello pondremos la postcondición {media > 0}.

```
Pre ≡ media > 0 [media ← (Σα:1 <= α <= N: v[α])/N] ≡

    (Σα:1 <= α <= N: v[α])/N > 0 ≡ Σα:1 <= α <= N: v[α] > 0
```


1.3. Utilización de asertos en lenguajes de prog.

- **Uso de asertos en Eiffel**
  - **En Eiffel, a la programación con asertos se la denomina programación por contrato.**
  - **especificación pre/post de función ≡ contrato**

    Si el programa que llama a la función cumple la precondición, la función se compromete a cumplir la postcondición.
  - **Eiffel permite los siguientes asertos:**
    - precondiciones (require)
    - postcondiciones (ensure)
    - Invariantes de bucle (invariant)
    - Función de cota de bucle (variant)
    - Invariante de clase (invariant)

1.3. Utilización de asertos en lenguajes de prog.

- **Ejemplo 1: División entera**

```
div(a, b: integer; resto: integer_ref): integer is
      require
         a >= 0 and b > 0
      do
         result := a // b
         resto.set_item (a \\ b)
         print(resto)
      ensure
         a = b * result + resto.item
         resto >= 0
         resto < b
      end
```

- **Ejemplo 2: Clase complejo en notación polar**


```
indexing
   title:       "Producto escalar de un vector de complejos";
   cluster:     "";
   project:     "";
   copyright:   "";
   author:      "Fernando Barber";
   original:    ;
   version:     1.0;
   last_change: no_changes;
   done_at:     "";
   extrnl_name: ""

class MAIN

creation
   make

feature {NONE} -- Creation

   make (args : ARRAY [STRING]) is
      local
         i: integer
         c1, c2, c3, res: complejo
      do
         print("Calculo de la suma de un vector de complejos.%N")
         create c1
         create c2
         create c3
         c1.set_rec(1,1)
         c2.set_rec(2,1)
         c3 := c1 + c2
         print(c3)

         io.get_char
      end -- make

end -- class MAIN
```

```
class complejo inherit math
feature
   mod: real
   arg: real
   re: real is
      do
         result := mod * cos(arg)
      end
   im: real is
      do
         result := mod * sin(arg)
      end
   set(m: real; a: real) is
      require  -- Para que el invariante se dispare
      do
         mod := m
         arg := a
      end

   set_rec(r: real; i: real) is
      require
      do
         mod := sqrt(r * r + i * i)
         if r = 0 and i = 0 then
            arg := 0
         else
            arg := arctan2(i, r)
         end
         if arg < 0 then
            arg := arg + 2 * pi
         end
      end
```

```
   infix "+"(c: complejo):complejo is
      do
         create result
         result.set_rec(re + c.re, im + c.im)
      end
```

Invariante de clase

```
   invariant
      modulo_positivo: mod >= 0
      arg >= 0 and arg < 2 * pi
end
```

1.3. Utilización de asertos en lenguajes de prog.

- **Uso de asertos en C/C++**
  - Únicamente existe la macro *assert(int test)* que se expande a una sentencia *if*. Si '*test*' es falso, el programa aborta.
  - Se tiene que incluir el fichero de cabecera <assert.h>
  - Si antes de incluir la librería se define la constante NDEBUG (#define NDEBUG), los asertos no tienen efecto.
  - Existen librerías no estándar para el uso de asertos. La más importante es Gnu Nana.

1.3. Utilización de asertos en lenguajes de prog.

- **Ejemplo: División entera**

```
// Descomentar el define para que los assert no se ejecuten
// #define NDEBUG
#include <assert.h>

int Div(int a, int b, int & resto)
{
    int c;

    // Pre:
    assert(a >= 0 && b > 0);

    c = a / b;
    resto = a % b;

    // Post:
    assert((a == b*c + resto) && (resto >= 0) && (resto < b) );

    return c;
}
```

1.3. Utilización de asertos en lenguajes de prog.

- **Como programar con asertos en C/C++**
  - Diferenciaremos entre versión de depuración y versión final.
  - Precondiciones e invariantes de clase:
    - Obligatorios en versión de depuración
    - Si la función es interna, la precondición se puede eliminar en la versión final.
    - Si la función es un interface a una estructura de datos o a un módulo de un programa, la precondición se ha de mantener como excepción.

1.3. Utilización de asertos en lenguajes de prog.

- Postcondiciones:
  - Opcionales en versión de depuración. Pueden ser únicamente un comentario.
  - Se puede comprobar únicamente parte de la postcondición y no toda completa.
  - Se eliminan en la versión final.
- Invariantes de bucle y funciones de cota:
  - Opcionales en versión de depuración. Pueden ser únicamente un comentario.
  - Se utilizan en algoritmos especialmente complicados o críticos.